SAVONIA

AMMATTIKORKEAKOULUTUTKINTO

TEKNIIKAN JA LIIKENTEEN ALA

MONINPELI UNITY PELI-MOOTTORILLA

TE-KIJÄ: Antti Pirskanen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Antti Pirskanen	
Työn nimi Moninpeli Unity-pelimoottorilla	
Päiväys 14.2.2018	Sivumäärä/Liitteet 27
Ohjaaja(t) Lehtori Jussi Koistinen	
Toimeksiantaja/Yhteistyökumppani(t) Savonia Ammattikorkeakoulu	

Tiivistelmä

Opinnäytetyön toteutettiin Savonia-ammattikorkeakoululle ja aiheena oli moninpeli Unity-pelimoottorille. Tarkoituksena oli luoda peli, jossa pelaajien olisi vuorotellen pyrittävä torjumaan kohti ammuttavaa palloa. Pelin tuli sisältää myös opetuksellinen komponentti. Lisäksi tuli toteuttaa selainpohjainen katselunäkymä, josta voitaisiin seurata pelien kokonaistilannetta

Opinnäytetyö aloitettiin suunnitelemalla annettujen vaatimuksien pohjalta itse peli ja siihen liittyvä selainsovellus sekä moninpeliominaisuuksien vaatima palvelinsovellus. Tekniikat rajoittuivat muutamiin vaihtoehtoihin esivaatimusten ja palvelinympäristön takia ja näistä valittiin sopivimmat.

Lopputuloksena saatiin vaatimusten mukainen, kaikki vaaditut komponentit sisältävä, toimiva moninpelikokonaisuus, jossa palvelinsovellus keskustelee Unity-pelin ja katselunäkymän kanssa websocket-yhteyden välityksellä. Pelin ulkoasuun jäi kuitenkin vielä paljon varaa jatkokehitykselle.

Avainsanat
Unity, ASP.NET, Websocket, C#, Moninpeli

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Antti Pirskanen			
Title of Thesis Multiplayer Game on Unity Game Engine			
Date	14 February 2018	Pages/Appendices	27
Supervisor(s) Mr. Jussi Koistinen, Senior Lecturer			
Client Organisation /Partners Savonia University of Applied Sciences			

Abstract

The purpose of this thesis was to create a multiplayer game on the Unity game engine for Savonia University of Applied Sciences. Thea aim was to create a game where a ball is shot towards players, who then have to try and deflect the ball. There was also a requirement for an educational component in the game as well as a webpage that would enable the user to view the overall situation in the concurrent active games.

The work began by planning the game and the webpage based on the set requirements as well as planning th e server-side application required by the multiplayer element. Available techniques were severely limited to only few alternatives by constraints caused by the prerequirements and server environment. From these options, the most suitable ones were selected: ASP.NET for the server-side application and AngularJS Javascript-framework for the webpage.

As a result of this thesis, a working system consisting of all the required components communicating via the web-socket connection was created. The graphics and user interface of the game remained in a state where they re-quire further development.

Keywords
Unity, ASP.NET, Websocket, C#, Multiplayer game

ESIPUHE

Haluan kiittää Savonia-ammattikorkeakoulun ohjelmistosuunnittelija Mikko Pääkköstä opinnäytetyön aikana saadusta avusta ja teknisestä tuesta. Lisäksi haluan kiittää Savonia-ammattikorkeakoulun lehtori Jussi Koistista opinnäytetyön ohjaamisesta.

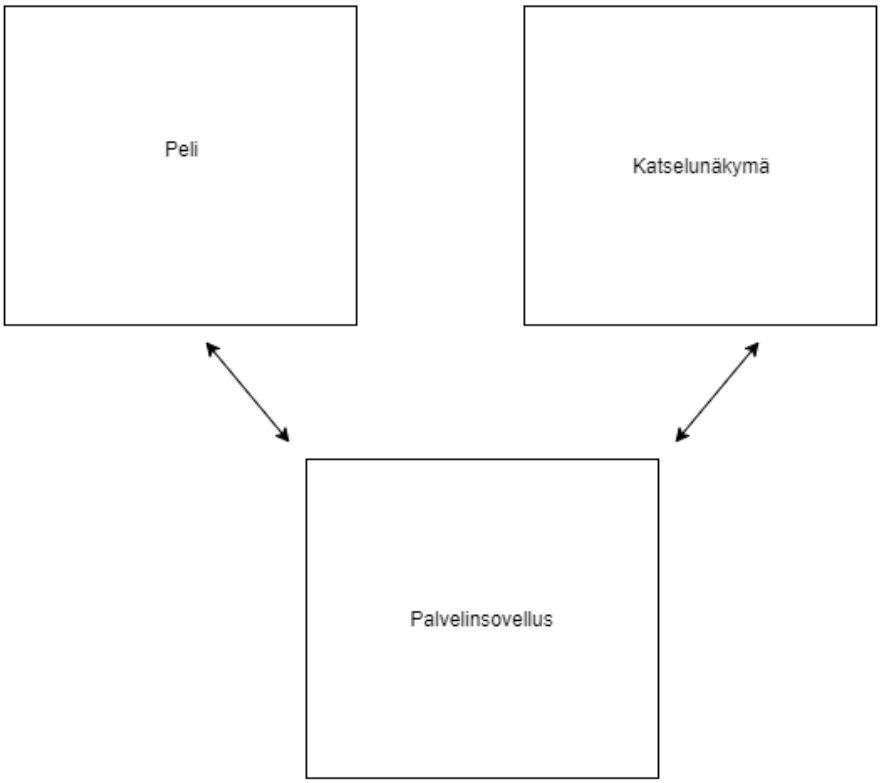
Kuopiossa 14.2.2018
Antti Pirskanen

SISÄLTÖ

1 JOHDANTO

Opinnäytetyön aiheena oli toteuttaa moninpeli Savonia-ammattikorkeakoululle. Peli tuli toteuttaa Unity-pelimoottorilla ja sen piti myös olla hyödynnettävissä jollain tavalla opetuskäytössä. Pelin lähtöajatuksena oli pelimekaniikka, jossa pelaajat yrittävät kukin omalla ruudullaan torjua palloa, joka torjumisen jälkeen siirtyy toiselle pelaajalle riippuen siitä, mihin kohtaa pelikenttää pallo torjunnan jälkeen osui. Lisäksi opinnätetyössä tuli toteuttaa web-selaimessa toimiva katselunäkymä, jolla voitaisiin tarkastella käynnissä olevan pelin kokonaistilannetta.

Näiden vaatimusten pohjalta opinnäytetyö jakautui kolmeen osaan, jotka näkyvät kuvassa 1: Unitylla toteutettavaan peliin, web-katselunäkymään ja näiden lisäksi palvelimella ajettavaan sovellukseen, jossa pelin suoritukseen liittyvät operaatiot varsinaisesti tapahtuvat ja johon aiemmin mainitut komponentit ovat yhteydessä. Peli ja katselunäkymä saavat palvelimelta tiedon käynnissä olevan pelin tilasta, jolloin ne osaavat päivittää näkymänsä vastaamaan palvelimelta saatua tilaa. Ne myös lähettävät tietoa palvelimelle pelaajan tekemistä toimista ja valinnoista, mitkä puolestaan taas vaikuttavat pelin tilaan palvelimella, kuvan 1 mukaisesti.



KUVA 1. Opinnäytetyön komponentit

2 SUUNNITTELU

2.1 Pelin suunnittelu

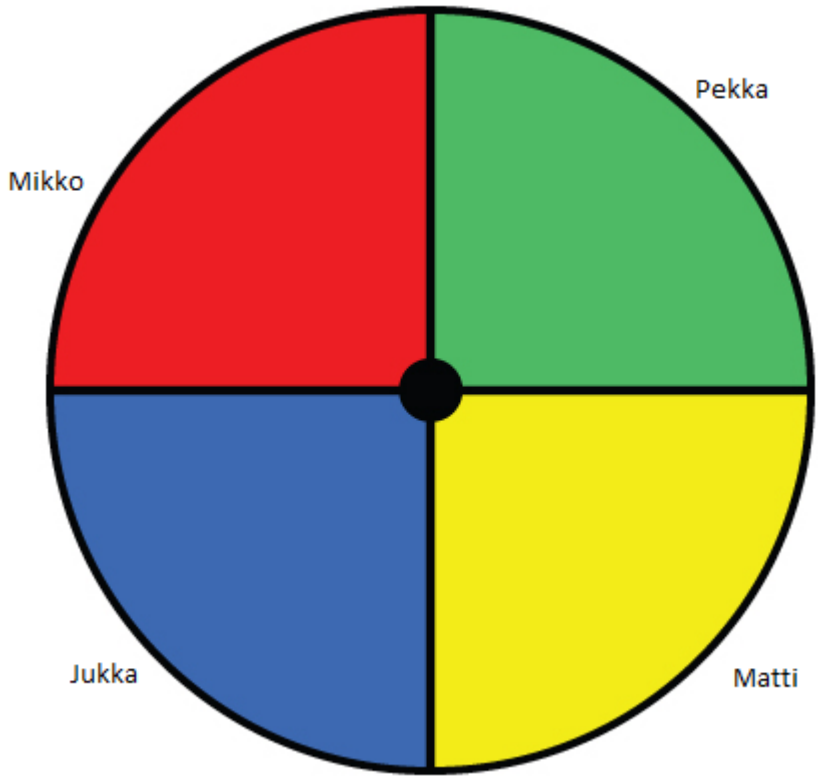
2.1.1 Pelin kulku

Palvelin suunniteltiin alusta alkaen tukemaan useampaa samanaikaista huonetta, joihin pelaajat voivat liittyä tai vaihtoehtoisesti luoda oman huoneensa. Huoneisiin suunniteltiin myös rajoitus, joka estää pelin alkamisen tai pysäyttää käynnissä olevan pelin, jos pelaajia on vain yksi.

Pelin kulku suunniteltiin aluksi seuraavanlaiseksi: Pelin alussa satunnaiselle pelaajalle lähetettiin pallo, joka ammutaan pelaajan pelikentän reunasta kohti keskustaa ja pelaajan oli osuttava omalla, hiirellä liikutettavalla lyöntipallolla kohtiammuttuun palloon. Riippumatta siitä osuiko pelaaja palloon vai ei, pallon osuessa pelikentän laitaan, se lähetetään taas eteenpäin satunnaisesti valitulle seuraavalle pelaajalle. Jos pelaaja ei osunut palloon, putosi hän pelistä pois.

Pelikentän suunniteltiin olevan kuvan 2 mukainen, jossa pelaaja voi liiikuttaa lyöntipalloaan hiirellä ympyränmuotisen kentän keskellä. Muiden pelaajien nimet on sijoitettu pelikentän reunoille ja kullekkin varattu yhtä suuri sektori ympyrästä. Pallon osuessa tietyn pelaajan sektoriin se lähetetään seuraavaksi tälle pelaajalle. Pelaajien pudotessa pelistä, pelaajat poistetaan kentän reunoila, jolloin kunkin pelaajan sektori suurenee.

Pelin kulku jalostui erämuotoiseksi, jossa pudonneet pelaajat putoavat pelistä erän ajaksi pois ja erän voittaja on viimeinen jäljellä oleva pelaaja, joka saa voitostaan yhden pisteen. Erille suunniteltiin myös huonetta luotaesssa määritettävä yläraja, jonka ylittämisen jälkeen peli loppuisi ja koko pelin voittaja olisi se kenellä olisi eniten erävoittoja. Tätä ominaisuutta ei kuitenkaan toteutettu ajanpuutteen vuoksi.



KUVA 2. Pelikentän suunnitelma

2.1.2 Pelillistäminen

Pelillistäminen tarkoittaa pelimäisten ja yleisemmin peleissä käytettävien elementtien ja toiminnallisuuksien hyödyntämistä jossain muussa yhteydessä, kuin varsinaisessa pelissä. Pelillistämisen tarkoituksena on muuttaa toiminta, jonka tekeminen koetaan tylsänä ja monotonisena, hauskemmaksi ja palkitsevammaksi peleistä tuttujen keinojen, kuten kokemustasojen, saavutuksien ja aikarajojen avulla. (Kapp 2012, 10-15.)

Pelin vaadittu opetuksen pelillistämisominaisuus päätettiin toteuttaa siten, että pelaajille esitetään pelaamisen ohessa kysymyksiä esimerkiksi ohjelmoinnista tai mistä tahansa muusta ylläpitäjän määrittämästä aiheesta. Kysymyksiin määritellään vastausvaihtoehdot, joista valitaan yksi numeronäppäimillä. Oikeista vastauksista ansaitaan kolikoita, jotka ilmestyvät pelikentän keskelle lisättyyn ympyränmuotoiselle puolustusalueelle. Pelaajan putoamista pelistä muutettiin tätä varten siten, että pelikentän reunalta ammuttu pallo ammutaan kohti puolustusalueella olevia kolikoita ja pallon osuessa kolikkoon, se häviää. Jos pelaaja menettää kaikki kolikkonsa, putoaa hän pelistä pois erän ajaksi.

2.2 Tekniikoiden valinta

Palvelimen ja pelin välisen kommunikaation toteuttamiseen oli kaksi vaihtoehtoa: Perinteinen http-yhteys, jossa pelin on jatkuvasti lähetettävä pyyntöjä palvelimelle, jotta pelin mahdollinen muuttunut tila saadaan tietoon tai vaihtoehtoisesti socket-yhteys, joka pysyy avoinna koko käyttäjän pelisession aja. Socket-tekniikka mahdollistaa datan lähettämisen myös palvelimelta asiakassovellukselle.

Sockettien nopeus, ja kahdensuurtainen tiedonsiirto tekivät niistä houkuttelevamman vaihtoehdon pelin ja serverin väliseen tiedonsiirtoon. Aluksi normaalit asiakasohjelmasta lähetetyt asynkroniset, sockettia hitaammat http-kutsut vaikuttivat riittävältä ratkaisulta pelin luonteesta johtuen: Jokainen pelaaja näkee vain oman liikkeensä omassa pelissään, joten esimerkiksi muiden pelaajien paikkatietoa ei tarvitse lähettää kaikille pelaajille useita kertoja sekunnissa. Toisaalta taas nopeampi socket-yhteys mahdollisti puolestaan sen, että odotusaika vuorojen välillä tippuu mahdollisimman pieneksi, mikä puolestaan taas poistaa turhaa odotteluaikaa pelaajilta. (Moskovits, Salim, Wang 2013, 1-2.)

Ongelmaksi kuitenkin muodostui se, että selainpohjaiset sovellukset, joiden kanssa palvelimen piti myös kommunikoida, eivät tue TCP-sockettia. Ratkaisu kuitenkin löytyi websocket-tekniikan muodossa, joka tarjoaa TCP-socketin kaltaisen nopean ja kaksisuuntaisen yhteyden selainsovelluksiin. Websocket ei kuitenkaan rajoitu pelkästään selaimiin, vaan sitä voidaan käyttää myös työpöytäsovelluksissa, mikä teki siitä ideaalivaihtoehdon, kun asiakasovelluksena oli perinteisempi työpöytäsovellus ja selaimessa toimiva sovellus. (Lombardi 2015, 1-7.)

Koska palvelinsovellusta oli tarkoitus ajaa Savonian palvelimilla, rajoittuivat sen toteutusvaihtoehdot lähinnä PHP ja ASP. NET – ratkaisuihin. Näistä kahdesta valittiin ASP. NET tilaajan yhteyshenkilön suosituksesta ja koska websocketin toteutus ASP. NET:llä vaikutti yksinkertaisemmalta. Palvelinpään ohjelmointikieleksi valittiin C#, koska se oli tekijälle jo hyvin tuttu.

Unity-pelin ohjelmointikieli rajoittui Unityn tukemiin C# ja Javascript kieliin, joista C# valittiin erityisesti sen takia, että palvelin toteutettiin samalla ohjelmointikielellä, joten tarvittaessa samaa koodia voitaisiin hyödyntää sekä pelin, että palvelinsovelluksen kehityksessä.

Selainpohjaisen katselunäkymän toteuttamiseen käytettyjen Javascript-kirjastojen vaihtoehtoina olivat jo aiemmin tutuksi JQuery ja AngularJS-kirjastot. Käytettäväksi kirjastoksi valittiin AngularJS, aiempien hyvien kokemusten ja kirjaston mahdollistaman nopean sovelluskehityksen takia. (AngularJS, 2017.) Lisäksi päätettiin käyttää Bootstrap-kirjaston valmiita tyylejä, siistin ulkoasun aikaansaamiseksi helposti.

3 TYÖKALUT JA TEKNIIKAT

3.1 Unity

3.1.1 Yleistä

Unity on Unity Technologiesin kehittämä alun perin vuonna 2005 julkaistu pelimoottori jolla voi kehittää kaksi- tai kolmiulotteisia pelejä. Unity tukee laajaa laitteiden kirjoa tietokoneista mobiililaitteisiin ja pelikonsoleihin. (Unity, 2016.) Unityn perusversiota, Personal editionia voi käyttää ilmaiseksi niin kauan kuin pelin mahdollisesti tuottama rahasumma on alle 100 000 dollaria. Maksullisissa versioissa ei tätä rajoitusta ole ja niissä on lisäksi kehittyneempiä lisäominaisuuksia, kuten parempia analytiikkaominaisuuksia. (Unity, 2016.)

3.1.2 Asset

Assetit ovat Unity-projektin rakennuspalikoita. Asset on mikä tahansa projektissa käytettävä esimerkiksi ääni-, kuva- tai 3d-mallitiedosto. Unityssa siis kaikki pelin tekemiseen käytetyt tiedostot ovat assetteja. (Goldstone 2009, 34).

3.1.3 Scene

Unityn scenet voidaan ajatella eri tasoina tai alueina pelissä. Scenejen käyttäminen hajauttaa latausaikoja ja mahdollistaa pelin eri osa-alueiden, kuten esimerkiksi tasojen yksittäisen testaamisen. (Goldstone 2009, 34-35.)

3.1.4 GameObject & Component

Kun asset sijoitetaan sceneen, siitä tehdään GameObject. Gameobject on Unityn perusosanen, joka voi olla esimerkiksi esine tai hahmo. Jokainen GameObject sisältää vähintään transform-komponentin, jota käytetään kertomaan pelimoottorilla GameObjectin sijainti, rotaatio ja skaala X, Y, Z koordinaatteina. Komponentit voivat vaikuttaa objektien käyttäytymiseen, ulkoasuun ja muihin objektin ominaisuuksiin. Gameobjectit ovatkin siis pohjimmiltaan kokoelma erilaisia komponentteja. (Goldstone 2009, 35.)

3.1.5 Prefab

Prefab on Unityn GameObject, joka on tallennettu assettiksi myöhempää käyttöä varten. Kun samaa objektia tarvitaan useammassa paikassa, siitä kannattaa tehdä Prefab. Tästä esimerkki on esimerkiksi vihollinen, jota kohdataan useita kertoja pelin aikana. (Goldstone 2009, 36.)

3.1.6 Skriptit

Gameobjectin käyttäytymistä voidaan ohjata siihen liitetyllä skriptikomponentilla. Skriptit voivat esimerkiksi laukaista tapahtumia, muokata muiden gameobjectiin liittyvien komponenttien ominaisuuksia, tai reagoida käyttäjän syötteeseen. Kuvassa 3 näkyvät start- ja update-metodit ovat vähimmäis-

vaatimus jokaiseen skriptiin. Start-metodia kutsutaan, kun skriptiin liittyvä gameobject luodaan, joten tässä metodissa voidaan tehdä peliobjektiin tarvittavat alustukset. Update-metodia kutsutaan jokaisella näytönpäivityksellä, jos gameobject on aktiivinen. Esimerkiksi update-metodissa voidaan ottaa kiinni käyttäjän syöte näppäimistöltä ja muuttaa gameobjectin sijaintia näytöllä. (Unity 2016.)

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

KUVA 3. Unity-skriptin perusrakenne

3.2 ASP. NET

ASP.NET on Microsoftin kehittämä web-ohjelmistokehys, jonka ensimmäinen versio julkaistiin vuonna 2002. Ohjelmistokehys on tarkoitettu etenkin dynaamisten web-sivujen ja sovellusten kehittämiseen. ASP.NET –sovelluksia voidaan kehittää C#-, Visual Basic.Net-, Jscript- ja J#-kielillä. (Tutorialspoint 2014.)

3.3 AngularJS

AngularJS on vuonna 2012 julkaistu Googlen ylläpitämä Javascript ohjelmistokehys, joka on tarkoitettu helpottamaan erityisesti yksisivuisten websovellusten kehitystä. AngularJS:n tarkoituksena on lisätä Javascript-pohjaisiin websovelluksiin tuki MVC-arkkitehtuurille, mikä mahdollistaa helpomman ja nopeamman kehityksen sekä testauksen. (Wahlin, 2013.)

3.4 JSON

JSON on lyhenne sanoista Javascript Object Notation ja se on avoin standardi tiedonvälitykseen. Opinnätetyössä olisi voinut käyttää yhtä hyvin jotakin toista standardia tiedonvälitykseen, kuten esimerkiksi XML. JSONin etuna on, että se on käytännössä javascript-objekti, joten sitä voidaan vaivattomasti käsitellä selainohjelmoinnissa ilman ylimääräisiä muunnoksia Javascriptin ymmärtämään muotoon. (Smith 2015, 37-38.)

3.5 Bootstrap

Bootstrap on avoimen lähdekoodin web-käytöliittymäkehys, joka sisältää valmiita tyylejä ja komponentteja. Bootstrap-kirjastoa hyödynnetään kehitettäessä responsiivisia, monelle eri näyttökoolle skaalautuvia websivuja ja – sovelluksia. (Rahman 2014, 8-9.)

3.6 Websocket

3.6.1 Yleistä

Websocket mahdollistaa nopean, kaksisuuntaisen kommunikaation sovelluksen ja palvelimen välillä. Vuonna 2010 websocket-tuki tuli ensin Google Chromeen ja jo vuonna 2011 se sai tuen myös mm. Firefox-, Safari, Internet Explorer, Opera-selaimilla. Palvelinpuolella websocket on tuettu mm. PHP, Java, Ruby, Node.js ja ASP.NET –ympäristöissä joihin on lisäksi tarjolla monia valmiita toteutuksia. (Moskovits, Salim, Wang 2013, 1-4).

3.6.2 Toimintaperiaate

Yhteys avataan lähettämällä ensin serverille http get-pyyntö, kuvan 4 mukaisilla otsikkotiedoilla, joissa kerrotaan pyynnön olevan websocket-pyyntö. Serverin hyväksyttyä pyynnön se palauttaa kutsuvalle sovellukselle kuvan 5 mukaisen vastauksen, jonka jälkeen websocket-yhteys aukeaa. (Mozilla Foundation 201.7)

```
1 GET /chat HTTP/1.1
2 Host: example.com:8000
3 Upgrade: websocket
4 Connection: Upgrade
5 Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
6 Sec-WebSocket-Version: 13
```

KUVA 4. Websocket-yhteyden avaamista edeltävä http-kutsu (Mozilla Foundation 2017.)

```
1 HTTP/1.1 101 Switching Protocols
2 Upgrade: websocket
3 Connection: Upgrade
4 Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
5
```

KUVA 5. Serverin vastaus hyväksyttyyn websocket-pyyntöön (Mozilla Foundation 2017.)

Kuvan 6 mukaisesti websocket-yhteys luodaan selaimen päässä Javascriptillä kutsumalla Websocket-olion konstruktoria, jolle annetaan parametriksi palvelimen osoite. Yhteyden muodostuttua, laukeaa onopen-tapahtumankäsittelijä. Onmessage-tapahtumankäsittelijä laukeaa aina dataa vastaanotettaessa ja onclose-tapahtumankäsittelijä laukeaa websocket-yhteyden katkettua. Dataa voidaan lähettää kutsumalla socket-olion send-funktiota, jonka parametriksi lähetettävä merkkijono annetaan. Nämä periaatteet ovat samat sekä palvelimen, että sovelluksen pään websocket-toteutuksessa riippumatta käytettävästä ohjelmointikielestä (Moskovits, Salim, Wang 2013, 6-9).

```
var host = "ws://localhost:8778/WSHandler.ashx";

socket = new WebSocket(host);

socket.onopen = function ()
{

}

socket.onmessage = function (msg)
{

}

socket.onclose = function ()
{
```

KUVA 6. Websocket-yhteys Javascriptillä

3.7 Visual Studio

Microsoft Visual Studio on Microsoftin ohjelmistokehitysympäristö, joka tukee useita eri ohjelmointkieliä, kuten esimerkiksi C#, C++ ja Visual Basic. Ympäristöllä voidaan kehittää esimerkiksi Windows Forms, ASP. NET tai Windows Phone – sovelluksia tai käyttää myös sellaisenaan pelkkänä tekstieditorina. Unity-editorin oletusskriptieditorina käytetään Visual Studiota, jonka uusin version tulee Unity-pelimoottorin asennuspaketin mukana. (Visual Studio, 2016.)

3.8 Blender

Blender on alun perin vuonna 1995 julkaistu avoimen lähdekoodin ilmainen 3D-grafiikan mallinnusohjelma. Blenderillä voi 3d-mallinnuksen lisäksi toteuttaa animaatioita ja jopa kokonaisia pelejä sisäänrakennetun Python skriptejä tukevan pelimoottorin ansiosta. (Blender.org, 2016.)

4 TOTEUTUS

4.1 Serveri

4.1.1 Websocket-yhteys

Palvelimen websocket-toteutukseen käytettiin Microsoftin omaa MicrosoftWebSockets-kirjastoa. Kuvassa 7 näkyvä WSHandler-luokka on ASP.NET:n geneerinen käsittelijä, jota serveriä käyttävät sovellukset kutsuvat yhteyden muodostusta varten. Jos vastaanotettu http-pyyntö on websocket-yhteyden avauspyyntö, instantioidaan uusi MicrosoftWebSockets luokan olio, joka sisältää tarvittavat tapahtumankäsittelijät ja metodit palvelimen ja asiakassovelluksen väliseen viestintään.

4.1.2 JSON

Tiedon välittäminen palvelimen ja sovellusten välillä tapahtuu JSON-muotoisella datalla. Kuvassa 1 esiintyvää JSONData-luokkaa käytetään luomaan haluttu data olioksi, jonka jälkeen luotu olio serialisoidaan JSON-merkkijonoksi Newtonsoftin JSON.NET kirjastolla. JSONData-luokkaa käytetään myös Unity-pelin kanssa, jossa serverin lähettämä data voidaan muuttaa helposti takaisin C#-olioksi samalla JSON.NET kirjastolla.

4.1.3 Pelaajan yhdistäminen ja nimen rekisteröinti

Uuden pelaajan yhdistäessä palvelimelle edellämainitulla tavalla, luodun Websocket-olion instanssissa oleva onopen-tapahtumankäsittelijä laukeaa, josta kutsutaan geneerisen GameManager-luokan addNewPlayer-funktiota, jolle annetaan parametriksi juuri kyseinen Websocket olio. Gamemanager luokassa instantioidaan uusi Player-luokan olio, jonka konstruktorille annetaan tässä vaiheessa vain juoksevasti numeroitu yksilöllinen id ja MicrosoftWebSockets olio. Luotu id lähetetään pelaajan peliin kutsumalla juuri luodun Player-olion MicrosoftWebSockets-jäsenmuuttujan send-metodia. Samalla pelaajalle lähetetään myös palvelimella sijaitsevasta tiedostosta luetut kysymykset ja vastaukset JSON-muodossa. Näiden lisäksi luotu Player-olio lisätään GameManager-luokan ylläpitämään listaan pelaajista, jotka eivät ole liittyneet mihinkään peliin.

Seuraavaksi serveri odottaa pelaajan rekisteröivän nimensä. Pelaajan lähettäessä dataa, kyseisen pelaajaolion MicrosoftWebSockets jäsenmuuttujan onmessage-tapahtumankäsittelijä laukeaa, josta kutsutaan GameManager-luokan dataHandler-funktiota, joka käsittelee vastaanotetun datan. Pelaajan lähettämässä JSON-muotoisessa datassa tulee olla aina status-kenttä, jotta tiedetään mihin toimintoon liittyen data lähetettiin ja pelaajan id, jonka avulla voidaan yhdistää toiminnot oikeaan pelaajaan ja peliin. Pelaajan lähettämä nimi asetetaan peliin liittymättömien pelaajien listalta etsittyyn Pelaaja-olion name-kenttään. Kun nimi on rekisteröity, pelaajalle lähetetään lista käynnissä olevista peleistä, jotka on säilötty GameManager luokassa olevaan listaan.

4.1.4 Pelin luominen ja peliin liittyminen

Peliä luotaessa peli lähettää palvelimelle oikean statuksen ja haluamansa nimen pelille. Data otetaan vastaan ja käsitellään edellä kuvatulla tavalla, jonka jälkeen kutsutaan GameManager luokan createGame-funktiota. Tässä funktiossa luodaan uusi Game-luokan instanssi, jolle annetaan parametreinä pelin nimi, juokseva id-numero ja pelin luoneen pelaajan Player-olio. Pelin luomisen jälkeen pelin luoneen pelaajan Player-olio poistetaan peliin liittymättömien pelaajien listalta, koska Game-luokan konstruktori lisää pelaajan luomaansa automaattisesti peliin. Juuri luotu Game olio lisätään GameManager-luokassa olevaan pelien listaan ja lisäksi luodaan uusi säie, jossa peli käynnistetään kutsumalla Game-luokan run-metodia. Käynnissä olevat säikeet lisätään myös omaan listaansa.

Käynnissä olevaan peliin voi myös liittyä joko pelaajana tai katsojana lähettämällä sovelluksesta palvelimelle liityttävän pelin id:n ja roolin, joka on joko pelaaja tai katsoja. Tällöin GameManager-luokassa etsitään käynnissäolevien pelien listalta annetun id:n mukainen peli ja pelaajan Player-olio lisätään joko pelaaja- tai katsojalistaan.

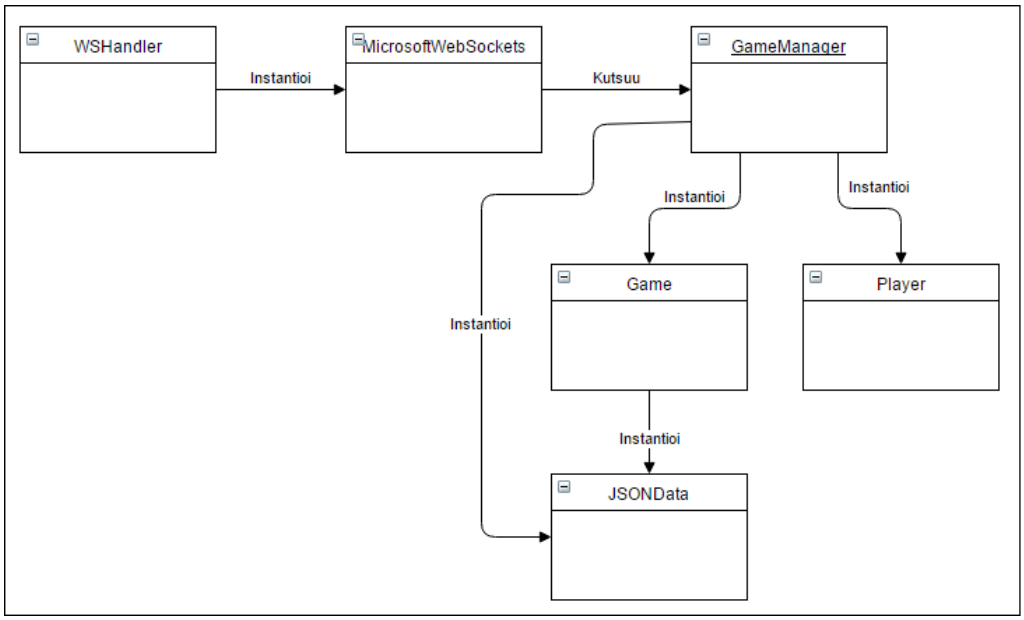
4.1.5 Peli-instanssi

Game-luokka sisältää pelin pääsilmukan, pelaajalistan, katsojalistan ja muut pelin tilaan liittyvät tiedot. Run-metodissa on koko peliolion elinajan pyörivä silmukka, jossa hoidetaan datan lähetys pelaajille ja katsojille. Silmukassa tarkistetaan sekunnin välein, onko pelin tila muuttunut, jolloin dataa lähetetään pelaajille ja katsojille vain, jos siihen on tarvetta. Lähetettävä data sisältää listan pelaajista, pelin sen hetkisen tilan (odottaa pelaajia, on alkamassa, peli käynnissä) ja muita tietoja kuten meneillään olevan erän ja pallon nopeuden. Data lähetetään kaikille pelaajille ja katsojille JSON-muodossa käymällä pelaaja- ja katsojalistat läpi ja kutsumalla kunkin Player olion Microsoft-WebSockets jäsenmuuttujan send-metodia.

Pallon siirto pelaajien välillä aloitetaan arpomalla ensimmäisellä kerralla pelaaja, jolle pallo lähetetään, jonka Player-olion hasball-kenttä asetetaan arvoon true. Tämän jälkeen kaikille pelaajille lähetetään pelin tilapäivitys, jonka jälkeen kyseisen pelin kohdalla jäädään odottamaan tilamuutosta. Kun pallon saanut pelaaja lähettää serverille takaisin dataa, on siinä seuraavan pallon vastaanottajan id sekä tieto putosiko pelaaja pelistä. Jos pelaaja putosi pelistä, asetetaan hänen Player-oliostaan dropped-jäsenmuuttuja arvoon true, jolloin hänelle ei enää kuluvan erän aikana voida lähettää palloa. Lisäksi tarkistetaan, oliko pelaaja toiseksi viimeinen pelissä mukana oleva pelaaja, jonka toteutuessa viimeisen pelaajan Player-olion points-jäsenmuuttujaan lisätään +1 ja aloitetaan uusi erä kasvattamalla Game-olion eränumeroa, palauttamalla kaikki pelaajat peliin mukaan ja arpomalla taas ensimmäisen pallon vastaanottaja.

Pelaaja poistetaan pelistä, kun hän on sulkenut websocket-yhteyden, mikä saadaan kiinni Microsoft-WebSockets-luokan onclose-tapahtumakäsittelijässä. Tapahtumakäsittelijästä kutsutaan GameManager luokan funktiota, joka huolehtii pelaajien poistamisesta. Funktiolle annetaan parametrina

kyseinen Websocket-olio, jonka avulla oikea pelaaja-olio etsitään ja poistetaan, joko ei peliin liittyineiden pelaajien listalta tai kunkin pelin pelaaja- tai katsojalistalta. Jos poistettava pelaaja on mukana parhaillaan käynnissä olevassa pelissä, täytyy tarkistaa myös, onko poistettava pelaaja viimeinen peliin vielä yhteydessä oleva pelaaja. Jos poistettava pelaaja on ainoa pelaaja pelissä, koko Game-olio tuhotaan: Gamemanager-luokan käynnissä olevien pelisäikeiden listalta etsitään, pysäytetään ja poistetaan tuhottavan pelin säie ja käynnissä olevian pelien listalta poistetaan pelin Game-olio.

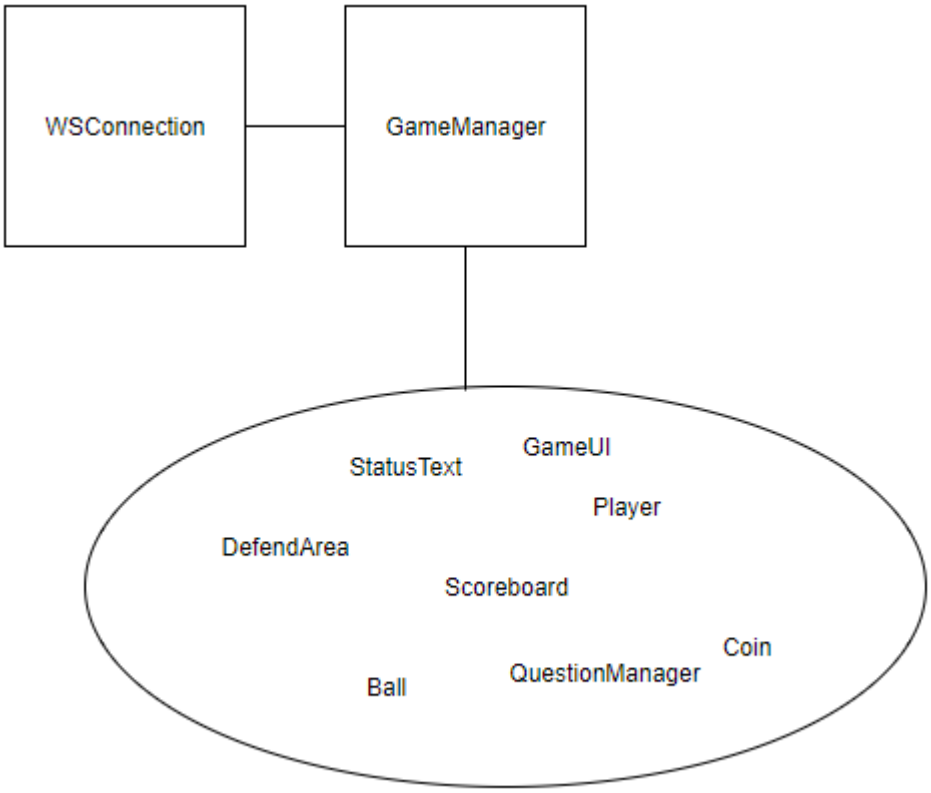


KUVA 7. Serverin arkkitehtuuri

4.2 Unity-peli

4.2.1 Arkkitehtuuri

Kuvassa 8 on esitetty pelin arkkitehtuuri, jonka keskiössä on GameManager: Tälle luokalle välitetään websocket-yhteydestä huolehtivalta WSConnection-luokalta palvelimen antama data, josta GameManager päättelee pelin sen hetkisen tilan ja piilottaa tai näyttää muut pelin elementit, kuten statustekstit käyttöliittymäkomponentit tai pelipallon. GameManager myös välittää tiedon palvelimelle WSConnection kautta, pelaajan tekemistä toimista, kuten esimerkiksi pallon osumisesta seinään, pelistä putoamisesta tai peliin liittymisestä.



KUVA 8. Pelin arkkitehtuuri

4.2.2 Websocket-yhteys

Websocket-yhteyteen Unityssa käytettiin GitHubista löytynyttä websocket-sharp–kirjastoa, joka sisältää korkean tason websocket implementaation C#-kielelle. Kirjasto saatiin toimimaan helposti pelin kanssa lataamalla kirjaston tiedostot sisältänyt paketti GitHubista, avaamalla pakettiin sisältyneen solution-tiedoston Visual Studiolla ja kääntämällä solutionin. Käännetyn solutionin Debug-kansiosta löytyvä .dll-tiedosto siirrettiin Unityn assetteihin, ja se lisättiin sceneen lisätyn tyhjän GameObjectin skriptiksi. Tyhjän gameobjectin käyttö mahdollistaa sen, että koodissa ei tarvitse erikseen huolehtia websocket-yhteyden avaamisesta, vaan yhteys avataan heti, kun tyhjä gameobject instantioidaan, mistä Unity huolehtii automaattisesti.

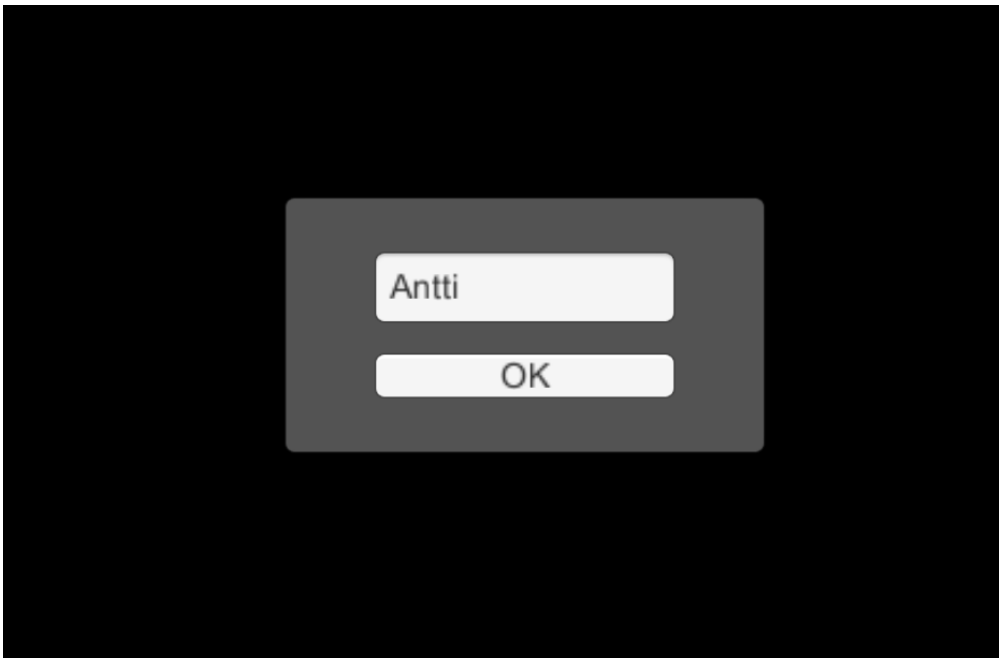
Tyhjän gameobjectin käyttäminen aiheutti kuitenkin ongelman scenestä toiseen siirtyessä: Edellisen scenen websocket-gameobjekti tuhottiin ja seuraavassa scenessä taas instantioitiin uudestaan, mikä aiheutti yhteyden katkeamisen ja uudelleenavauksen. Unityssa on Objekteille DontDestroyOnLoad-funktio, joka estää objektin tuhoamisen scenen vaihdossa, mutta syystä tai toisesta tätä ei saatu toimimaan, joten ongelma kierrettiin käyttämällä pelissä vain yhtä sceneä. Koska peli on melko pieni, eli siinä ei käytetä paljon gameobjekteja, ei yhden skenen käyttäminen aiheuttanut ongelmia esimerkiksi latausajoissa.

Websocket-serverin osoite asetettiin luettavaksi pelin kansiossa olevasta .cfg-tiedostosta, jotta sitä on helppo muuttaa tarvitsematta muokata itse koodia.

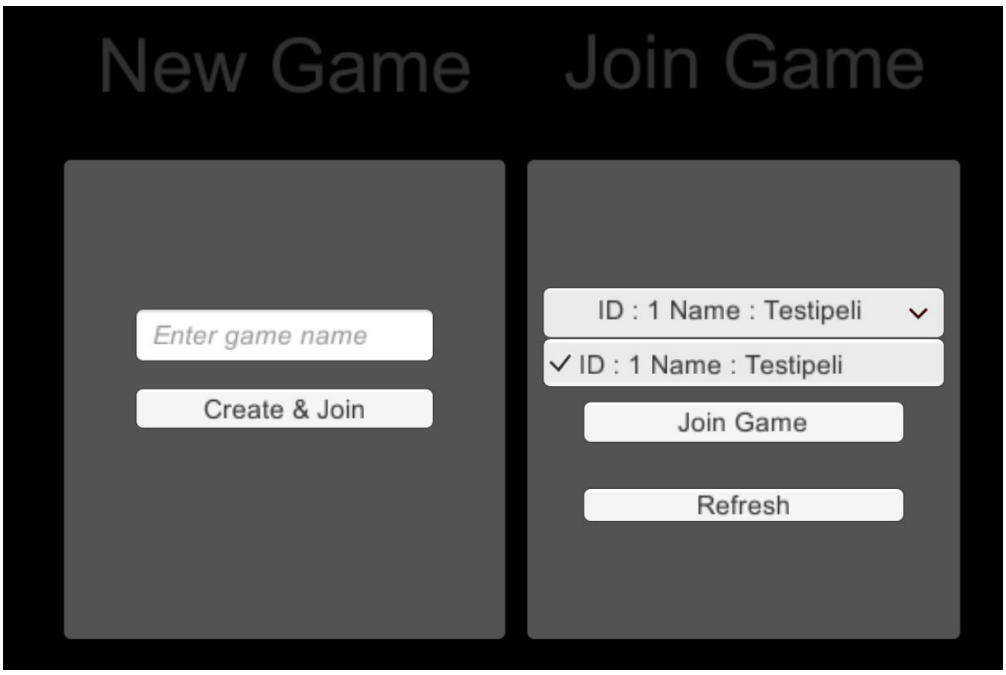
4.2.3 Alkunäkymät

Alkunäkymät käsittävät nimenkysymis-näkymän ja pelin luomis- ja peliinliittymisnäkymän. Valikot oli tarkoitus toteuttaa ensin omiin sceneihinsä, mutta edellämainittujen syiden takia ne sijoitettiin samaan sceneen, kuin itse pelinäkymä. Valikot toteutettiin Unityn canvas-objekteina, joihin lisättiin Unityn omia käyttäliittymäelementtejä kuten tekstikenttiä ja painikkeita.

Kuvassa 9 esiintyvä näkymä on ensimmäinen pelaajalle pelin käynnistyksen jälkeen näytettävä näkymä ja siinä kysytään vain pelaajan nimi. Pelaaja syöttää nimensä tekstikenttään ja painaa OK-nappia, jolloin serverille lähetetään tieto pelaajan nimen rekisteröinnistä, jonka jälkeen serveri lähettää vastauksena listan käynnistä olevista peleistä. Samalla pelissä nimivalikko piilotetaan ja kuvassa 10 esitetty pelin luomis- ja peliinliittymisnäkymä asetetaan näkyväksi. Tässä näkymässä pelaaja voi joko luoda uuden pelin tai liittyä olemassa olevaan huoneeseen valitsemalla pudotusvallikosta pelihuoneen. Kun käyttäjä on joko liittynyt peliin tai luonut uuden pelin, siitä lähetetään tieto serverille ja serverin lähetettyä vastauksen liittymisestä, luomis- ja liittymisnäkymä piilotetaan, jolloin näkyviin tulee varsinainen pelinäkymä.



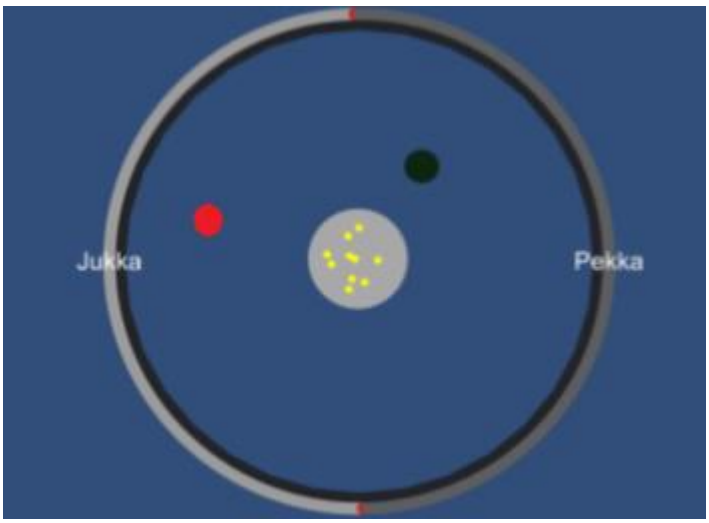
KUVA 9. Nimenkysymisnäkymä



KUVA 10. Pelin luomis- ja peliinliittymis-näkymä

4.2.4 Pelialue

Kuvassa 11 olevan pelinäkymän ympyränmuotoinen pelialueen ulkoreuna luotiin Blenderillä, koska Unitysta ei löytynyt valmiina tarvittavaa onttoa ympyrää tai lieriötä. Blenderissä luotu 3d-malli saatiin helposti siirrettyä Unityyn tallentamalla malli Blenderissä ja siitämällä syntynyt .blend-tiedosto Unityn assetteihin, josta sitä voitiin käyttää sellaisenaan pelissä. Vastapelaajien nimet sijoitettiin pelikentän reunoille laskemalla jokaiselle pelaajalle samankokoinen sektori ympyrästä ja laskemalla nimen sisältäneille gameobjecteille paikkavektori rotaatiomatriisilla: Rotaatiomatriisilla voidaan laskea vektorin komponentit, kun tunnetaan toinen vektori ja tiedetään tunnetun ja tuntemattoman vektorin välinen kulma. Tuntematon vektori lasketaan funktiossa kuvan 12 mukaisesti, jossa x on tunnetun vektorin x-komponentti, y on tunnetun vektorin y-komponentti ja alpha vektoreiden välinen kulma. Tunnettuna vektorina käytettiin sceneen sijoitettua näkymättömän gameobjectin paikkavektoria, joka sijaitsee ympyrän kaarella kello kahdessatoista. (Weissten, 2003.)



KUVA 11. Pelialue

```
float newx = x * Mathf.Cos(alpha) + y * Mathf.Sin(alpha);
float newy = -x * Mathf.Sin(alpha) + y * Mathf.Cos(alpha);

return new Vector3(newx, newy, 0);
```

KUVA 12. Vektorin laskeminen kulman avulla

Lisäksi, jotta pelaajien sektorit olisi helpompi erottaa toisistaan, tuli pelikentän laidan värin vuorotella pelaajien sektorien välillä siten että vierekkäiset sektorit eivät olisi samanväriset. Blender mahdollisti usean eri tekstuurin lisäämisen samaan 3d-malliin, mutta sektoreiden dynaamisen luonteen takia tätä ratkaisua ei voinut käyttää, koska sektorien koko ja määrä riippui pelaajien määrästä. Ongelma ratkaistiin sijoittamalla pelikentän reunoille pieniä pallonmuotoisia gameobjecteja vieri viereen, jotka saavat aikaan illuusion yhtenäisestä viivasta. Pallot sijoitettiin kaarelle yhden asteen välein ja niiden paikkavektorit laskettiin myös rotaatiomatriisilla.

Aivan pelikentän keskellä kuvassa 11 sijaitsevan puolustusalueen harmaa osa on tausta, jonka tarkoitus on toimia puolustusalueen referenssinä ja josta lasketaan kolikoiden paikat. Kunkin kolikon paikkavektori arvotaan sitä luotaessa siten, että kolikko jää harmaan alueen sisäpuolelle.

4.2.5 Pelaajan pallo

Pelaajan hiirellä liikutettava pallo on ohjelmoitu seuraamaan hiiren kursoria, mistä ongelmaksi koitui se, että pallon liikkumista ei saanut rajoitettua pelikentän sisälle tai puolustusalueen ulkopuolelle. Jotta pelaaja ei voisi suojella puolustusalueen kolikoitaan vain viemällä palloa niiden päälle, pallon mesh disabloidaan kokonaan, jos pallon y ja x koordinaatit ovat puolustusalueen rajojen sisällä. Tämä muutttaa pallon näkymättömäksi ja lisäksi poistaa sen fysiikanmallinnuksen, jolloin pelaajan pallolla ei ole vaikutusta torjuttavaan palloon. Sama toiminnallisuus toteutettiin pelaajan liikuttaessa

hiiren pelikentän ulkopuolelle. Tällä tavoin pelaajaa ohjataan hienovaraisesti pitämään pallo vain sallitulla alueella.

4.2.6 Torjuttava pallo

Torjuttava, punainen pallo asetetaan oletuksena poistetuksi scenestä disabloimalla sen meshrenderkomponentti. Kun serveri lähettää tiedon kyseiselle pelaajalle pallon saamisesta, pallon paikkavektori muutetaan vastaamaan edellisen pallonhaltijan sektorin keskikohtaa pelikentän reunalle kuvan 13 mukaisesti. Lisäksi, jotta pallo osuu aina varmasti ainakin yhteen puolustusalueen kolikkoon, etsitään scenen kaikki kolikot coin-tagilla ja näistä arvotaan yksi kohteeksi. Kolikon paikkavektorista vähennetään torjuttavan pallon paikkavektori ja näin saatu vektori normalisoidaan, jolloin saadaan arvottua kolikkoa kohti osoittava vektori. Tämä vektori asetetaan torjuttavan pallon nopeusvektoriksi ja kerrotaan lisäksi serverin antamalla, jokaisella vuoronvaihdolla kasvavalla nopeusarvolla. Jokaiseen kolikkoon on liitetty skripti, jonka tehtävänä on tunnistaa torjuttavan pallon osuma kolikkoon ja tuhota kolikko.

Torjuttavan pallon osumia laitoihin tai pelaajan palloon ei tarvitse erikseen laskea, vaan Unityn fysiikkamoottori huolehtii niistä automaattisesti ja muuttaa pallon suuntaa ja nopeutta realistisesti. Kun pallo osuu joko torjunnan jälkeen tai suoraan pelikentän laitaan, tarkistetaan ensin pelaajan omien kolikoiden määrä ja kenen pelaajan sektorille pallo osui. Serverille lähetetään tieto pallon seuraavasta vastaanottajasta ja mahdollinen tieto pelaajan putoamisesta erästä, jos pelaajan kolikot loppuivat. Pallon meshrenderer-komponentti disabloidaan, jolloin se palaa taas näkymättömäksi.

```
if (ballReceived && !receivedExecuted)
{
    gameObject.transform.position = VectorCalc.getVectorFromAngle(playerAngle) / 1.5f;
    Vector3 v;
    System.Random rnd = new System.Random();
    int idx = rnd.Next(0, GameObject.FindGameObjectsWithTag("coin").Length);

    v = GameObject.FindGameObjectsWithTag("coin")[idx].transform.position;

    v = (v - transform.position).normalized;

    rb.velocity = v*(initSpeed + speedInc);

    receivedExecuted = true;
}
```

KUVA 13. Pallon paikan, suunnan ja nopeuden asettaminen

4.2.7 Kysymykset ja tuloslista

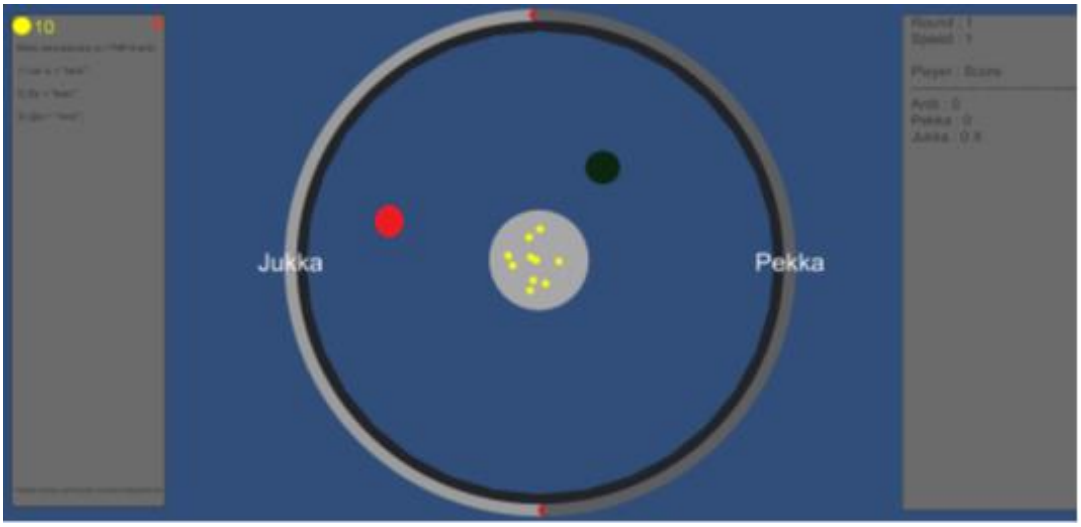
Kysymykset ladataan palvelimelta JSON muodossa nimen rekisteröimisen yhteydessä ja saatu data deserialisoidaan C#-oliolistaksi Newtonsoftin JSON.NET kirjastolla. Pelin alkaessa kysymyksistä arvotaan näytettäväksi yksi ja itse kysymys ja sen vastausvaihtoehdot näytetään kuvan 14 vasemmassa reunassa olevassa paneelissa. Tässä paneelissa näytetään lisäksi myös oikeista vastauksista ansait-

tujen kolikkojen määrä ja jäljellä oleva kysymyksen vastausaika. Jokaiseen kysymykseen on vastausaikaa 10 sekuntia, jonka jälkeen kysymys poistetaan näkyviltä ja arvotaan uusi kysymys. Kysymyksiin vastataan valitsemalla oikea vastaus numeronäppäimillä. Jos vastaus on oikea välähtää paneeli vihreänä ja pelaajan puolustusalueelle lisätään uusi kolikko. Jos vastaus on puolestaan väärä, välähtää paneeli punaisena.

Kuvassa 14 oikealla näkyvällä tuloslistalla näytetään tekstikomponentissa tämän hetkisen erän numero, pallon sen hetkinen nopeus ja huoneessa olevat pelaajat ja näiden pistemäärät. Pelaaja merkitään lisäksi X-kirjaimella, jos pallo on hänen pelissään.

4.2.8 Ilmoitukset

Pelin tilasta näytetään ilmoituksia pelinäkymän päällä olevassa textmesh-komponentissa, joka näytetään ja piilotetaan tarpeen mukaan. Pelin tila päätellään serverin lähettämästä datasta ja esimerkiksi kuvassa 15 on ilmoitus lisäpelaajien odottamisesta. Muita ilmoituksia ovat pelaajan putoaminen pelistä ja varoitus pelin alkamisesta.



KUVA 14. Pelinäkymä

KUVA 15. Esimerkki ilmoituksesta

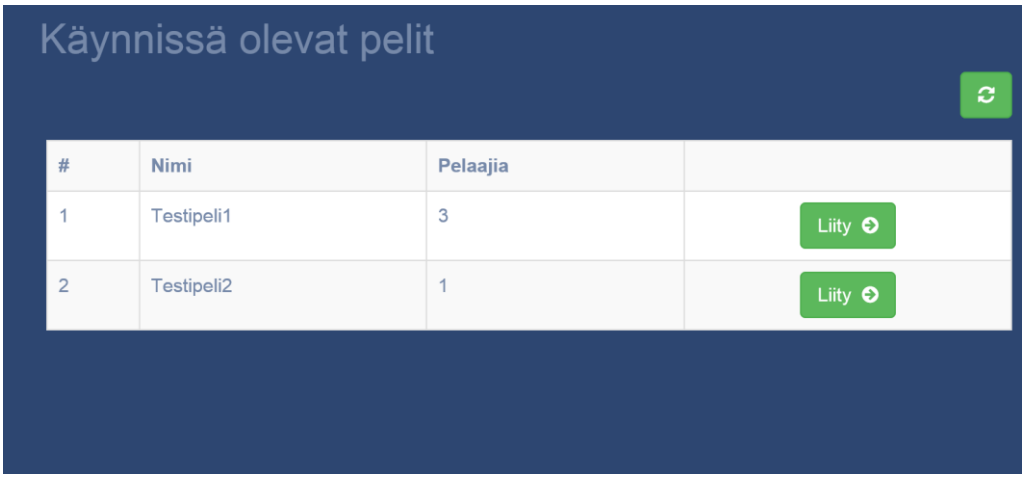
4.3 Pelien web-katselunäkymä

Web-näkymä toteutettiin yhtenä web-sivuna, jossa on lista käynnissä olevista katseltavista peleistä ja itse pelinkatselunäkymä. Elementtejä näytetään ja piilotetaan tilanteesta riippuen siten, että vain joko lista tai katselunäkymä on yhdellä kertaa näkyvillä. Sivun tyyleissä käytettiin pääosin bootstrapin valmiita tyylejä.

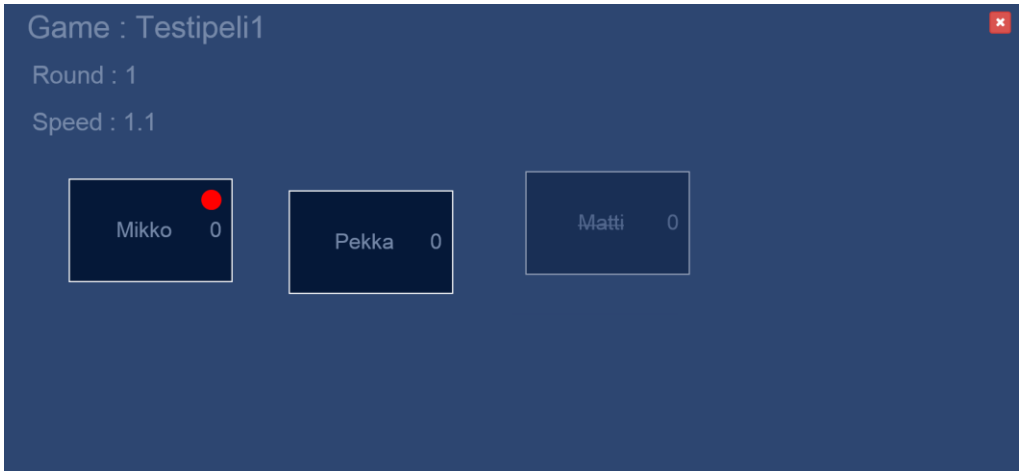
Websocket-yhteyteen käytettiin Javascriptin omaa websocket-toteutusta. Sivua avattaessa websocketyhteys muodostetaan ja käyttäjä rekisteröidään automaattisesti palvelimelle, jonka jälkeen serveri palauttaa käynnissä olevat pelit.

Kuvassa 16 esiintyvään pelinvalintanäkymään rakennetaan AngularJS:llä palvelimelta saadusta datasta html-taulukko, jossa pelin nimi, pelaajamäärät ja painike pelin katselun aloittamiseksi.
Pelin katselunäkymässä (kuva 17), näkyy pelin tietoja ja pelin pelaajat siirreltävinä laatikoina, jotka sisältävät pelaajan nimen, pistemäärän ja pallokuvakkeen, jos pallo on kyseisen pelaajan pelissä. Lisäksi erästä pudonneiden pelaajien laatikot on himmennetty.

Ajatuksena on se, että pelaajat voi hiirellä raahamalla asettaa esimerkiksi luokkahuoneessa pelattaessa istumajärjestyksen mukaisesti. Drag and drop ominaisuus toteutettiin Angular drag and drop-kirjastolla, joka on JQueryUI:n drag and drop ominaisuuksien toteutus AngularJS:ssä. AngularJS aiheutti ongelman, jossa koko listan päivittäminen uuteen palvelimelta saatuun pelaajalistaan aiheutti raahattujen elementtien palaamisen takaisin alkuperäisille paikoilleen. Ongelma ratkaistiin funktiolla, jossa ei korvata alkuperäistä mallina toiminutta listaa vaan vertailllaan alkuperäistä ja uutta, serveriltä saatua listaa ja tehdään tarvittaessa yksittäisiin pelaajaobjekteihin muutoksia.



KUVA 16. Katselunäkymän pelilista



KUVA 17. Valitun pelin katselunäkymä

5 YHTEENVETO

Opinnäytetyön tuloksena saatiin Unity-pelimoottorilla opetyskäytössä hyödynnettävä toimiva moninpeliprototyyppi. Opinnäytetyön aikana opittiin Unity-pelinkehityksen perusteet ja ASP.NET serverin toteutus. Websocket-palvelin onnistui erityisen hyvin arkkitehtuurinsa osalta, joka voitiin huomata opinnäytetyön tekemisen aikana muutoksien ja uusien ominaisuuksien tekemisen helppoutena.

Itse pelistä saatiin myös toimiva sovellus, mutta sen arkkitehtuurinrakenne jäi hieman sekaiseksi, mikä tulee varmasti vaikeuttamaan mahdollista jatkokehitystä. Asia olisi voitu korjata heti kättelyssä perehtymällä paremmin oppaisiin, jossa käsiteltiin Unity-pelien arkkitehtuuria ja hyviä kehitysperiaattteita. Työn tekemisen ajoittainen katkonaisuus aiheutti sen, että pitkien taukojen välillä varsinkin Unityn asiat unohtuivat, jolloin aikaa meni asioiden uudelleen opetteluun, joten lopputulos olisi ollut varmasti parempi, jos työtä olisi tehty enemmän yhteen menoon.

Myös pelin ulkoasu jäi melko keskeneräiseksi osaksi ajanpuutteen ja osaksi tekijän puuttellisten graafisten taitojen takia. Pelin perusideasta johtuen opinnäytetyössä ei ehkä saatu kaikkea hyötyä irti Unity-pelimoottorista: Koska peli oli käyttöliitymältään ja toiminnoiltaan melko yksinkertainen, sen olisi voinut toteuttaa helposti myös esimerkiksi selainpohjaisesti.

Suurin opinnäytetyön jatkokehityskohde on siis pelin ulkoasun suunnittelu ja toteutus. Opinnäytetyöstä opituilla tekniikoilla voisi myös tehdä monimutkaisemmankin moninpelin, jossa pelaajat voisivat vaikuttaa toisiinsa suoraan samalla pelialueella. Websocket-tekniikan reaaliaikaisuus mahdollistaisi pelin, jossa pelaajat voisivat nähdä toisensa liikkuvan pelissä reaaliajassa auttaen tai vaikeuttaen toistensa pelaamista ja näin lisäten pelaajien välistä interaktiota.

Opinnäytetyön tekemisen aikana heräsi myös ajatuksia opetuksen pelillistämisen kehittämisestä, niin että pelillistäminen ei rajoittuisi vain luokkahuoneseen ja muutamaan oppiaineeseen, vaan esimerkiksi koko opintoajan käytettävässä sovelluksessa voisi seurata omaa opinnoissa edistymistään, suorittaa viikottaisia haasteita ja tavoitteita sekä ansaita pisteitä ja mitaleita.

LÄHTEET JA TUOTETUT AINEISTOT

KAPP, Karl 2012. The Gamification of Learning and Instruction. San Francisco: Pfeiffer

WEISSTEIN, Eric W. Rotation Matrix, 2003 MathWorld A Wolfram Web Resource. [Verkkojulkaisu]. [Viitattu 2017-11-23.] Saatavissa: http://mathworld.wolfram.com/RotationMatrix.html

BLENDER. Features [Verkkojulkaisu]. [Viitattu 2017-10-9.] Saatavissa: https://www.blender.org/features/

VISUAL STUDIO. Visual Studio features [Verkkojulkaisu]. [Viitattu 2017-10-10.] Saatavissa: https://www.visualstudio.com/vs/features/

RAHMAN, S. F. 2014. Jump Start Bootstrap. Collingwood, Victoria: SitePoint.

TUTORIALSPOINT. 2014. ASP.NET Tutorial. 2014. [Verkkojulkaisu]. [Viitattu 2017-11-12.] Saatavissa: https://www.tutorialspoint.com/asp.net/asp.net_tutorial.pdf

WAHLIN, D. 2013. AngualrJS in 60 minutes. [Verkkojulkaisu]. Wahling Consulting. [Viitattu 2017-11-11.] Saatavissa: http://fastandfluid.com/publicdownloads/AngularJSIn60MinutesIsh_DanWahlin_May2013.pdf

MOSKOVITS, Peter ja SALIM, Frank ja WANG Vanessa. 2013. The Definitive Guide to HTML5 Websocket. New York City: Apress.

LOMBARDI, Andrew. 2015. Websocket: Lightweight Client-Server Communications. Sebastopol: O'Reilly Media Inc.

MOZILLA FOUNDATION. 2017. Writing Websocket Servers. [Verkkojulkaisu]. [Viitattu 2017-11-30.] Saatavissa: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers

GOLDSTONE, Will 2009. Unity Game Development Essentials. Birmingham: Packt Publishing.

SMITH, Ben 2015. Beginning JSON. New York: Apress

ANGULARJS. 2017. Why AngularJS. [Verkkojulkaisu]. [Viitattu 2018-1-12.] Saatavissa: https://angularjs.org

UNITY. 2016. Made with Unity. [Verkkojulkaisu]. [Viitattu 2018-1-4.] Saatavissa: https://made-with.unity.com

UNITY. 2016. Creating and Using Scripts. [Verkkojulkaisu]. [Viitattu 2018-1-12.] Saatavissa: https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html